ARL-TN-0673 ● MAY 2015

US Army Research Laboratory

A Communication Protocol for CyAMS and the Cyber Fighter Associate Interface

by David Harman, Scott Brown, Brian Henz, and Lisa M Marvel

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-0673 ● MAY 2015

US Army Research Laboratory

A Communication Protocol for CyAMS and the Cyber Fighter Associate Interface

by David Harman
College Qualified Leaders Student, University of Maryland, College Park

Scott Brown
Secure Mission Solutions

Brian Henz and Lisa M Marvel
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
May 2015	Technical Note	June 2014–January 2015

4. TITLE AND SUBTITLE
A Communication Protocol for CyAMS and the Cyber Fighter Associate Interface

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
David Harman, Scott Brown, Brian Henz, and Lisa M Marvel

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIN-D
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0673

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In the future, cyber attacks will have a growing effect on the outcome of military operations. The Cyber Fighter Associate (CyFiA) is a decision support program that works with the cyber network modeling and simulation system "CyAMS". The CyFiA will allow network security personnel to keep mission integrity through the use of cyber agility maneuvers. The CyFiA is being tested on CyAMS, a tool that uses an ns-3-based large network simulator. A large of amount of information needs to be exchanged between these 2 programs, and an efficient protocol and method for communication is required. A communication protocol utilizing specific ports is described in this report. It is this protocol that enables the CyFiA programs to communicate and operate in large network simulations.

15. SUBJECT TERMS
cyber modeling, simulation, cyber security, software patch management, tactical networks

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Lisa M Marvel
Unclassified	Unclassified	Unclassified	UU	28	19b. TELEPHONE NUMBER (Include area code) 410-278-6508

Contents

List of Figures

Preface

This report documents a portion of a larger project. There are 2 related works that were done in concert. The first is the *Cyber Fighter Associate*, which is currently in press with the US Army Research Laboratory (ARL).[1] The second is *Cost Computations for Cyber Fighter Associate*, also documented in an ARL technical note, ARL-TN-0674.[2]

[1] Huber C, Marvel LM. Cyber fighter associate. Aberdeen Proving Ground (MD): Army Research Laboratory (US); in press.

[2] Erbs A, Marvel LM. Cost computations for cyber fighter associate. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2015 May. Report No.: ARL-TN-0674.

INTENTIONALLY LEFT BLANK.

1. Introduction and Background

As the military has adopted more networked equipment, the opportunity for cyber attacks to occur has also risen. To mitigate the effect these attacks have on a mission, network administrators and security experts must be able to decide on the best course of action based on many factors. The Cyber Fighter Associate (CyFiA) will help decide the best course of action given a set of cyber agility maneuvers by measuring the cost and utility of potential maneuvers, along with node and network facts, to selecting the maneuvers that will lead to mission success.

The idea of the CyFiA comes from a line of decision-making programs starting with the Pilot Associate. The Pilot Associate was a system created to assist pilots in making decisions quickly (Smith 2004). The Warfighter Associate (Buchler et al. 2013) is a more recent program that is based on the same concepts as the Pilot Associate. It is meant to assist military decision making on the ground in a variety of different scenarios. Both of these programs rely on "concepts such as cognitive workload, multi-tasking, task completion timing, force synchronization, information sharing, and decision quality" (Buchler et al. 2013). However, the main overarching concept is the Observe, Orient, Decide, and Act (OODA) loop. The OODA loop is a military decision-making process that allows personnel to quickly and effectively respond to a situation. Currently under development, the CyFiA is a program based on the concepts used in the aforementioned programs. Unlike previous programs, however, the CyFiA is meant to analyze networks, apply agility maneuvers to accomplish a mission, and make decisions to preserve their integrity.

The CyFiA will suggest agility maneuvers to accomplish a mission in response to a vulnerability or threat (known infection). The first mission for the CyFiA is to evaluate agility maneuvers and to provide recommendations for a patch management mission. These maneuvers are a set of various actions that can be used to prevent the propagation of an attack by patching a device so it is immune to the attack. The CyFiA is meant to be expandable to allow for new agility maneuvers to be added in the future.

To test the effectiveness of the CyFiA, we are leveraging a program called Cyber Army Modeling and Simulation (CyAMS), which makes use of the ns-3 network simulator, a discrete-event network simulator for Internet systems (ns-3 2004). CyAMS can model very large-scale networks with the help of a high-performance-computing system. Currently, CyAMS is implemented on a system called Thufir. Thurfir is a hybrid computer mixing graphics-processing unit (GPU) and standards cores. It has 6,576 total compute cores. CyAMS has demonstrated the ability to

model networks containing up to 35 million nodes, so using CyAMS on Thufir will be a very effective test bed for the CyFiA.

To accomplish agility maneuver simulations for the large simulated networks, there needs to be a method to transfer the considerable amount of data from CyAMS to the CyFiA knowledge engine. Therefore, we designed and developed a communications program that transfers the data between the separate programs.

The overall objective of this portion of the project is to define a protocol for communication between the different programs and create a testing program to confirm successful operation. To enable this overarching objective, we needed to use an efficient and flexible means of communication.

2. Design

The CyFiA currently consists of 3 parts: the CyFiA knowledge-based system, the Risk-Cost Calculation program, and CyAMS. The network is being simulated on CyAMS. The Risk-Cost Calculation program needs input from CyAMS to calculate cost/utility, and they provide information to the CyFiA knowledge-based engine so it can recommend agility maneuvers to CyAMS. There needs to be frequent communication between the programs, so we defined a protocol to do this. A protocol is a set of formalized rules that explains how data are communicated over a network. Our protocol relies on the exchange of User Datagram Protocol (UDP) packets utilizing specific ports. UDP is a minimal message-oriented Transport Layer Protocol (protocol is documented in IETF RFC 768 [Postel 1980]) that allows for efficient message passing between programs and computers.

To support the CyFiA, the following information for each node needs to be exchanged between the programs:

- Location (latitude/longitude)
- Capability and operating system
- Node health information
- Edge endpoint and communication throughput
- Battery information
- State change (agility maneuver, health, etc.)
- Patch size
- Graphical user interface (GUI) update information (state change, GUI information)

As the CyFiA project continues, additional information will need to be shared, so we designed an expandable communication protocol to enable this sharing.

2.1 Protocol

Our expandable protocol was designed to allow a high data transfer rate as well as the ability to add functionality to the programs in the future. Figure 1 depicts a visual representation of the communication exchange protocol. Each program is represented as a table with a blue heading. We selected port numbers starting at Port 3010. Starting from the left side of the graphic, we can see that CyAMS provides various inputs to the communications program. More specifically, it sends data using ports starting at 3010 incremented by 10 (3010 receives decisions from the CyFiA, and 3020 is not used for consistency purposes). Ports in the 3000 range were chosen because they were not commonly used by other programs in our environment. The communications program distributes information between different programs. As the communications program receives information from CyAMS, it resends the information to the CyFiA knowledge tool and Risk-Cost Analysis program. Although we are running these programs on the same machine for testing purposes, sockets do not permit listening and sending on the same port. Therefore, whichever port CyAMS sends on, the communications program will resend on that port, plus 1 for CyFiA or plus 2 for Risk-Cost (e.g., CyAMS sends on 3040, and CyFiA would receive on 3041). There are 2 ports that are not consistent with the rest of the system. Port 3082 is used by CyFiA to receive data about patch sizes from Risk-Cost, and Risk-Cost sends patch sizes to the GUI on port 4000. Each port that sends information has its own protocol for the information that is sent in each packet, as shown in Fig. 1. Every sending port has 2 things in common. The first is the request ID, which is a number that accompanies each packet so that the programs can keep track of the information. The second piece is the node ID, which is the identifying number of a specific node on the CyAMS graph. Besides these common pieces of information, each port sends different pieces of information. An attempt was made to keep similar pieces of information together for usability purposes. When the programs are expanded in the future, the protocol will allow new ports to be added with ease.

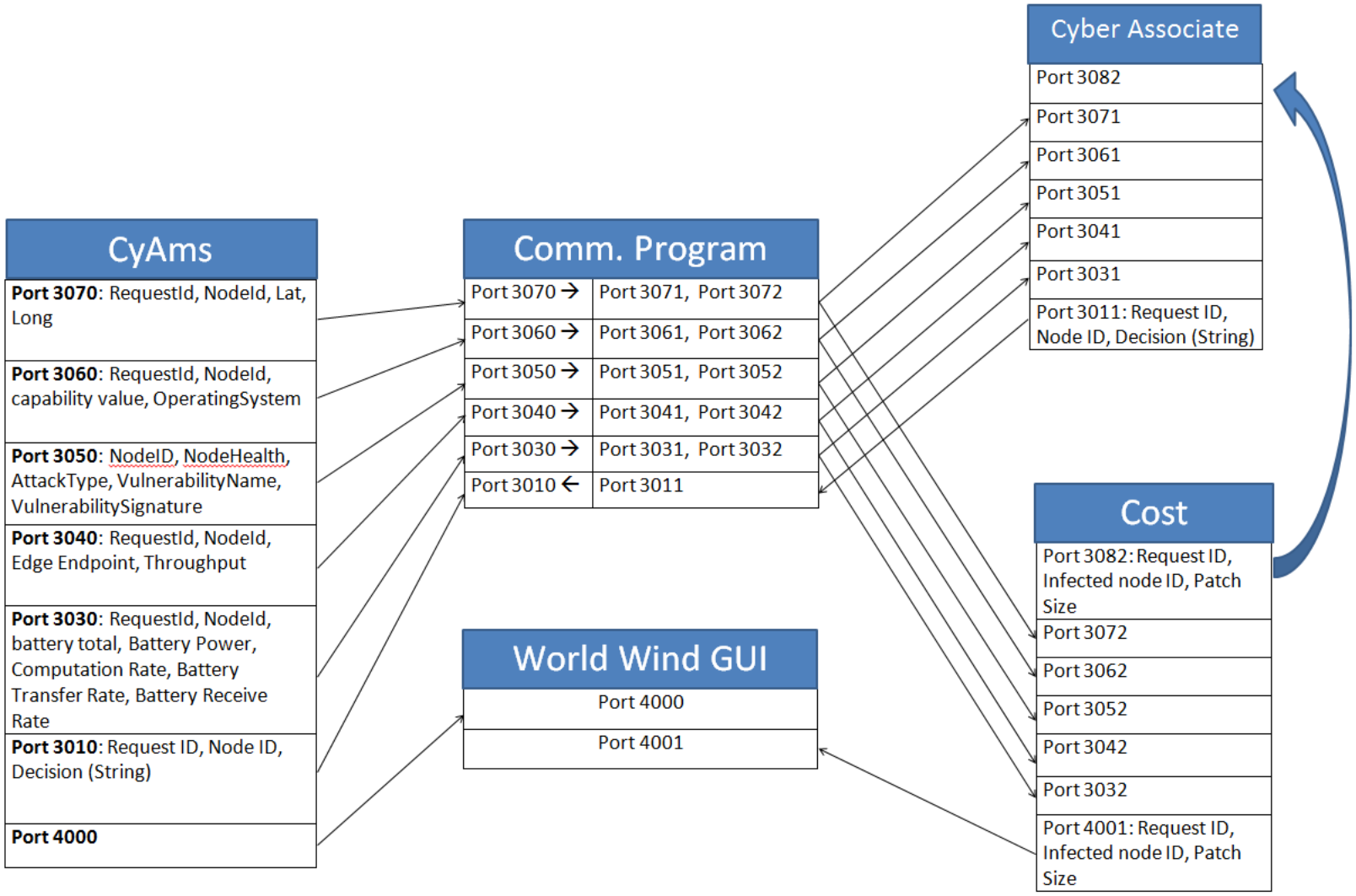

Fig. 1 Model showing the port protocol between the programs

Note that the World Wind GUI is one of the programs that receives information using this protocol. The World Wind GUI receives state updates from the CyAMS simulation any time a node within the simulation changes state. In addition to node state changes, the GUI also receives any data regarding link changes that may occur as a result of the simulation or due to a critical path change. These state changes will then be reflected within the GUI itself. This can be seen in Fig. 2. The links highlighted in yellow represent the critical path nodes that are required for the mission: the green node is patched or immune and the red node is the source of the infection.

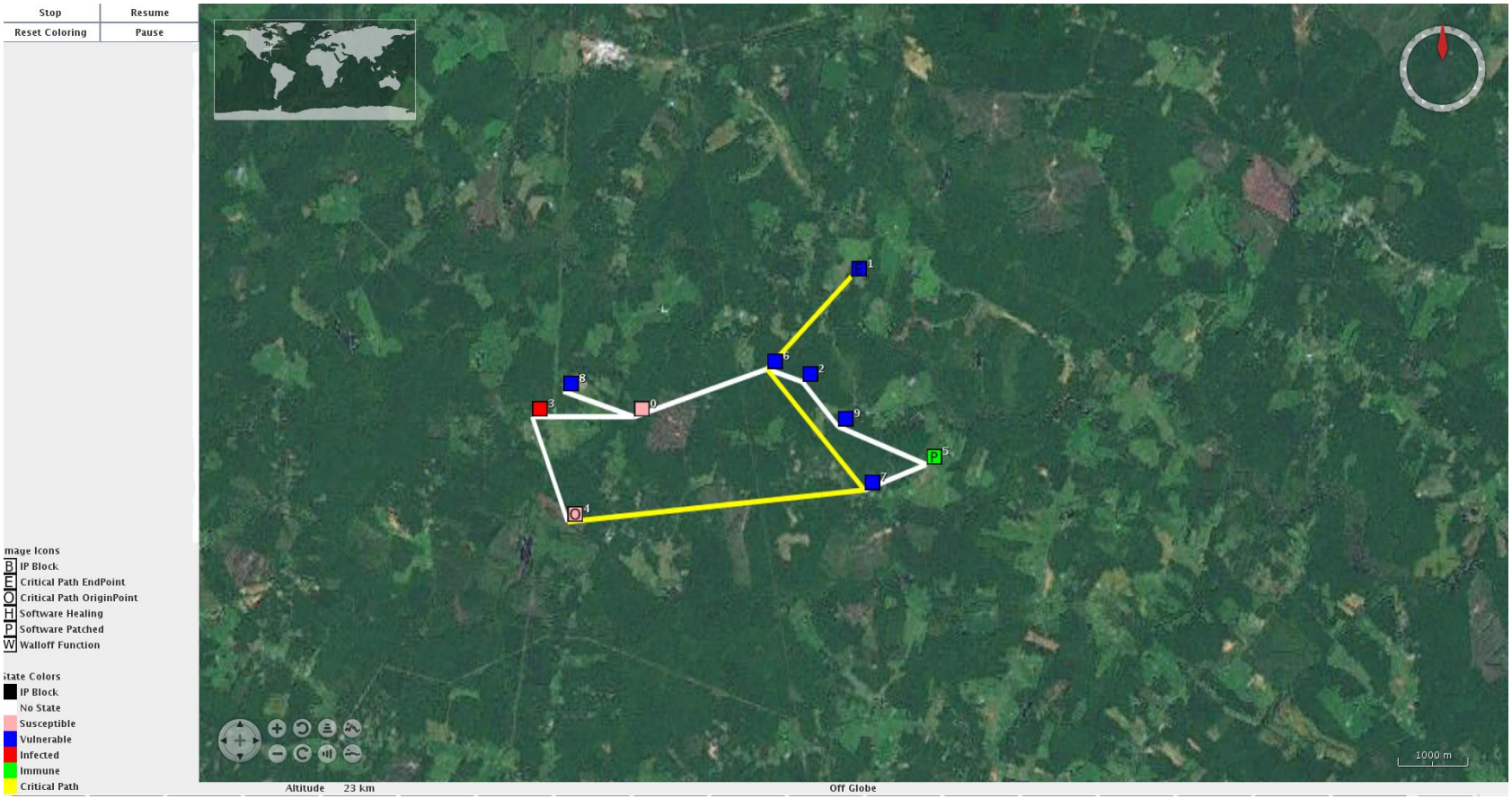

Fig. 2 World Wind GUI

2.2 Communications Program Output

The communications program is the main program that connects the CyFiA knowledge base program, Risk-Cost program, and CyAMS together. In the communications program, each port uses its own thread so that packets are not skipped. After the program receives each packet, it resends the packet to both the CyFiA and the Risk-Cost program. This is repeated with packets being received from the CyFiA and Risk-Cost programs. Figure 3 depicts a window showing data sent through the communications program by CyAMS.

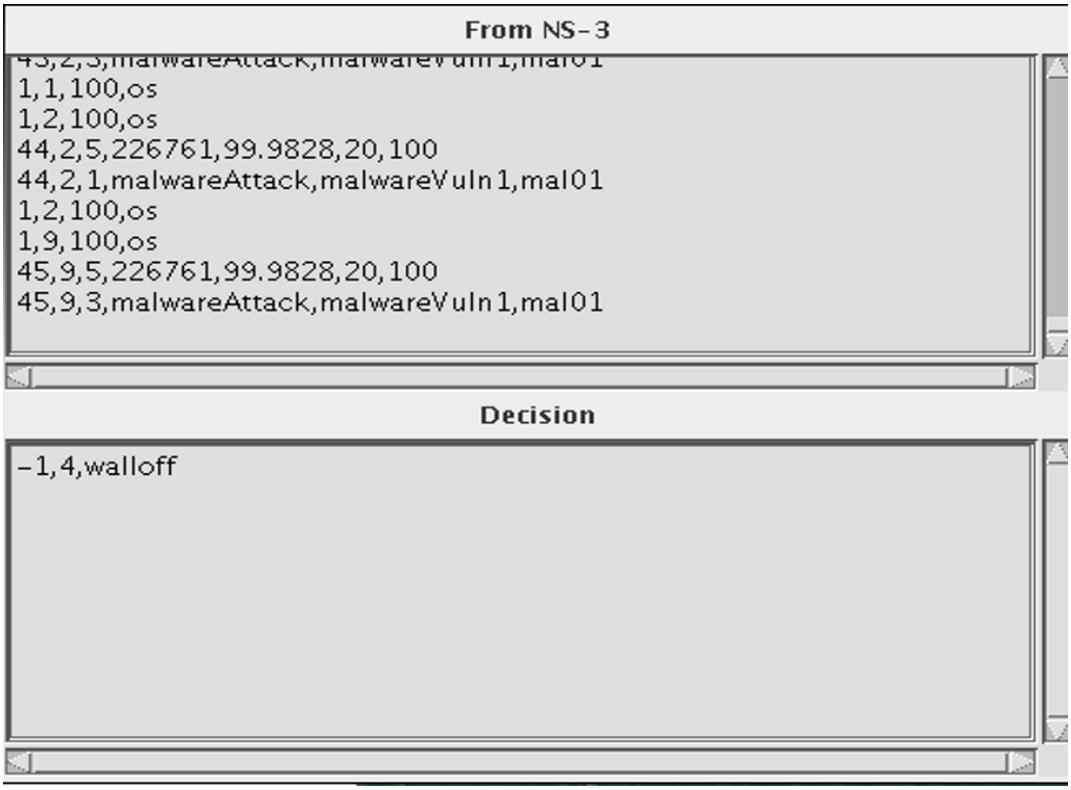

Fig. 3 Program modeling data being received by the Cyber Associate and Risk-Cost program

Figure 4 shows a window program used to test the communications program. The upper table represents CyAMS and the different packets it will send. The lower box represents either the CyFiA knowledge or the Risk-Cost program and the data it would receive from CyAMS.

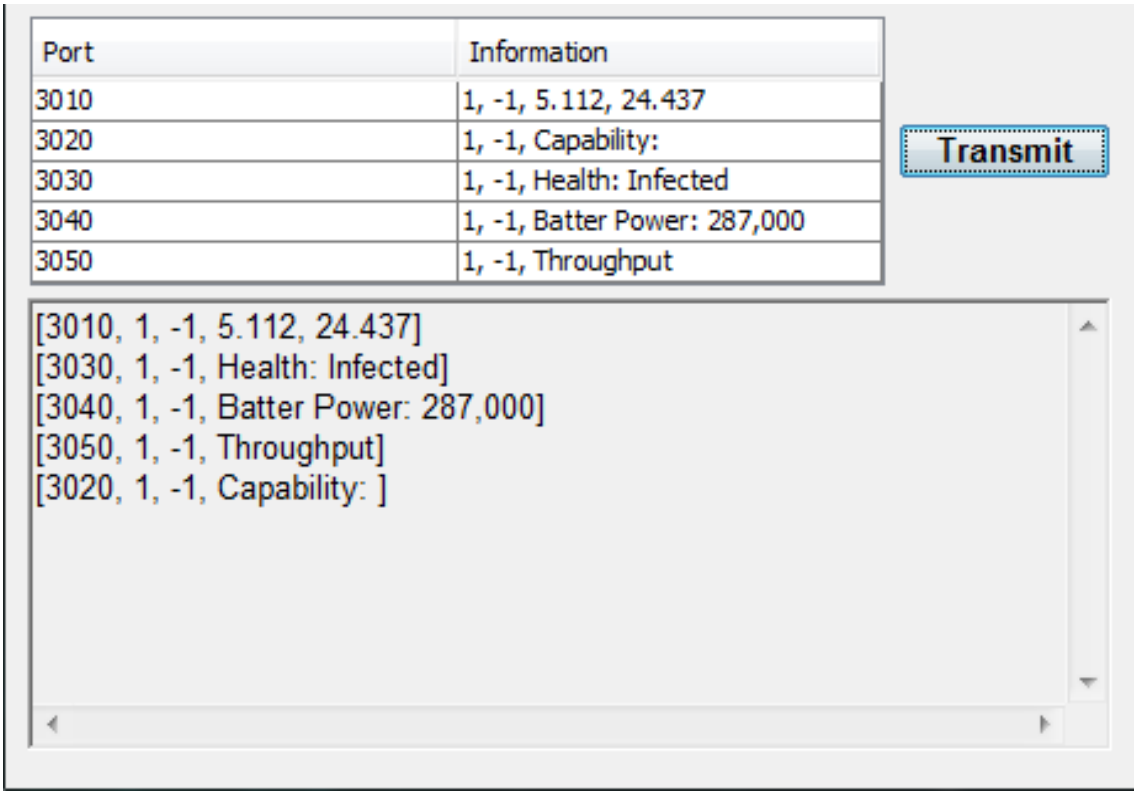

Fig. 4 Test demonstrating the communications program sending and receiving information

2.3 Program Flow

Figure 5 shows the general flow of the program from start to finish. The program will be started by the GUI, which is based on NASA's World Wind (Maxwell 2004). The GUI is directly connected to CyAMS via port 4000, as seen in Fig. 1. CyAMS will then begin to send data packets to the CyFiA and Cost program. Using this data, the Cost program will begin to construct an onboard representation of the network. After the construction data are sent, the CyFiA will begin to make decisions based on the network/node conditions. When the CyAMS network is updated, it will send out an update to the Risk-Cost program to add to the onboard graph. If there are no updates and the simulation is finished, the program will end. Otherwise, it will wait until there is an update or the program is ended. A code listing is provided in the Appendix.

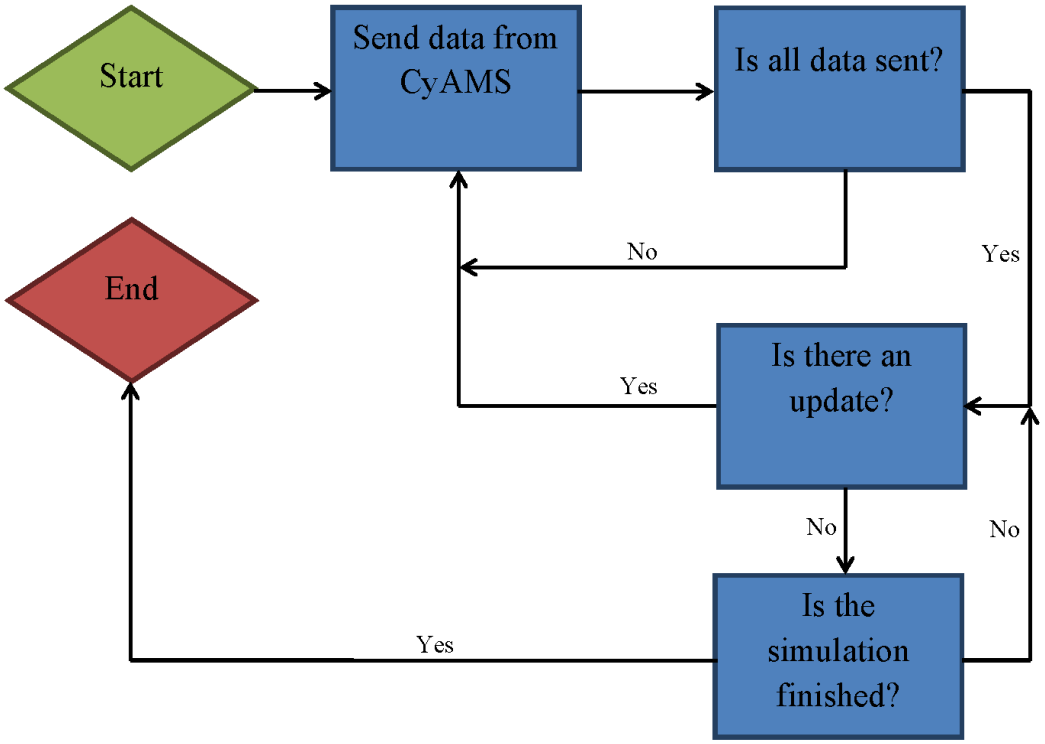

Fig. 5 Flowchart showing the progression of the program

3. Conclusion and Future Work

The field of cyber defense is rapidly expanding. To effectively counter cyber attacks, we have to be able to react quickly and respond correctly. The CyFiA will help to evaluate the cost and utility of different agility maneuvers that may be employed within networks. As a proof of concept, we used a network simulator called CyAMS. Using this network simulator, we will be able to obtain a course of action for cyber agility maneuvers given a multitude of situations. To get these programs to interact, we designed a communication protocol and program that uses ports to exchange UDP packets. The communication protocol is easily modified and can quickly transmit information.

Although the general layout of communication is complete, the communication program will have to be modified when the CyFiA is extended. Another change that can be made is the integration of the communication program into the CyFiA. At the current state of the CyFiA, we wanted to keep the overall program more modular. However, as it becomes more refined, the communication program can be directly added into the CyFiA.

Although not directly related to the communication protocol of the program, a main GUI could also be added to the program. This GUI would allow the user to more easily interface with the CyFiA by looking at things such as a network map or a list of outputs. Although the CyFiA is meant to aid a human operator, having a user-friendly interface will make more effective use of the system output.

We would also like to note that while Ports 3010–4001 work fine in our environment, they are previously designated for use by other programs/devices in open forums. For instance, Port 3070 is designated for mgxswitch communication devices, and Port 3010 is specified for the Telerate Workstation communication (Admin Subnet 2015). In the future, the programs/protocol will be modified to use ephemeral ports (i.e., ports 49152 through 65535) to avoid collision with previously specified port designations. Ephemeral ports are temporary ports assigned by a machine's IP stack and are assigned from a designated range of ports for this purpose.

4. References

Admin Subnet: TCP/UDF port finder [updated 2014 Aug 26; accessed 2015 Feb 2]. http://www.adminsub.net/tcp-udp-port-finder/3070.

Buchler N, Marusich L, Bakdash J, Sokoloff S, Hamm R. The warfighter associate: objective and automated metrics for mission command. Paper presented at: 18th International Command and Control Research and Technology Symposium; 2013 Jun 19–21; Alexandria, VA.

Maxwell C, Kim R, Gaskins T, Lam B, Hogan P. World wind. Moffett Field (CA): NASA Ames Research Center; 2004 [accessed 2014 Aug 4]. http://worldwind.arc.nasa.gov/features.html, 2004.

ns-3. Documentation; c2015 [accessed 4 Aug 2014]. http://www.nsnam.org /documentation/.

Postel J. Internet Engineering Task Force (IETF): RFC 768, User Datagram Protocol; 1980 Aug 28 [accessed 2015 Apr 28]. https://www.ietf.org/rfc/rfc768.txt.

Smith D. The Pilot's Associate Program; 2004 [accessed 4 Aug 2014]. http://www.dms489.com/PA/PA_index.html.

INTENTIONALLY LEFT BLANK.

Appendix. Code Listing

This appendix appears in its original form, without editorial change.

ThreadClass.java (main)

```
import java.util.ArrayList;


public class ThreadClass {
		public static void main(String[] args) {
			ArrayList <Thread> threads = new ArrayList <Thread>();

			//Decision <-
			Sending port3010 = new Sending(3011, 3010);
			Thread thread3010 = new Thread(port3010);
			threads.add(thread3010);

			//Critical Path <-
			Sending port3020 = new Sending(3020, 3022);
			Thread thread3020 = new Thread(port3020);
			threads.add(thread3020);

			//Node Info ->
			Sending port3030 = new Sending(3030, 3031, 3032);
			Thread thread3030 = new Thread(port3030);
			threads.add(thread3030);

			//Throughput ->
			Sending port3040 = new Sending(3040, 3041, 3042);
			Thread thread3040 = new Thread(port3040);
			threads.add(thread3040);

			//Node Health ->
			Sending port3050 = new Sending(3050, 3051, 3052);
			Thread thread3050 = new Thread(port3050);
			threads.add(thread3050);

			//capability->
			Sending port3060 = new Sending(3060, 3061, 3062);
			Thread thread3060 = new Thread(port3060);
			threads.add(thread3060);

			//Location ->
			Sending port3070 = new Sending(3070, 3071, 3072);
			Thread thread3070 = new Thread(port3070);
			threads.add(thread3070);

			for(Thread i: threads){
				i.start();
			}
```

```
                    for(Thread i: threads){
                        try {
                            i.join();
                        } catch (InterruptedException e) {
                        }
                    }

                }
}
```

Sending.java

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class Sending implements Runnable{
        int receiving, sending1, sending2;
        //receiving from one port sending to two
        public Sending(int receiving, int sending1, int sending2){
            this.receiving = receiving;
            this.sending1 = sending1;
            this.sending2 = sending2;
        }

        //receiving from one port sending to one
        public Sending(int receiving, int sending1){
            this.receiving = receiving;
            this.sending1 = sending1;
            this.sending2 = 0;
        }

        //Call this method to run each instance
        public void Run() throws Exception{
            DatagramSocket serverSocket = new DatagramSocket(re-
ceiving);

            while(true)
            {
                byte[] receiveData = new byte[1024];
                byte[] sendData = new byte[1024];
                //receives packet
                DatagramPacket receivePacket = new Datagram-
Packet(receiveData, receiveData.length);
                serverSocket.receive(receivePacket);
                //for testing purposes
```

```
                        String inData = new String( receivePacket.get-
Data());
                        //System.out.println(inData);

                        //sets the IP Address to that of this machine
                        InetAddress IPAddress = InetAddress.getBy-
Name("localhost");
                        //just for clarity purposes
                        sendData = receiveData;
                        //makes a new Datagram and sends it
                        DatagramPacket sendPacket =
                                new DatagramPacket(sendData,
sendData.length, IPAddress, sending1);
                        serverSocket.send(sendPacket);
                        //if the second port is present, it sends it to that one.
                        if(sending2 > 0){
                                DatagramPacket sendPacket2 =
                                        new Datagram-
Packet(sendData, sendData.length, IPAddress, sending2);
                                serverSocket.send(sendPacket2);
                        }
                }
        }

        //Runnables run method calls Run method
        public void run() {
                try {
                        this.Run();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Receiving.java

```
package CyFi_and_Cost_analysis;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class Receiving implements Runnable{
        int receiving;
        int sending1;
        byte[] sendData = new byte[1024];
```

```
		public Receiving(int receiving){
			this.receiving = receiving;
		}

		public void Run() throws Exception
		{
			DatagramSocket serverSocket = new DatagramSocket(re-
ceiving);
			byte[] receiveData = new byte[1024];
			while(true)
			{
				//receives packet
				DatagramPacket receivePacket = new Datagram-
Packet(receiveData, receiveData.length);
				serverSocket.receive(receivePacket);
				//for testing purposes
				String inData = new String( receivePacket.get-
Data());
				System.out.println(inData);

				//*****Sends to the Decision Port*****
				String out = "Received Decision";
				sendData = out.getBytes();
				//sets the IP Address to that of this machine
				InetAddress IPAddress = InetAddress.getBy-
Name("localhost");
				//makes a new Datagram and sends it
				DatagramPacket sendPacket =
						new DatagramPacket(sendData,
sendData.length, IPAddress, 3011);
				serverSocket.send(sendPacket);
				//if the second port is present, it sends it to that one.

				//*****Sends to the Request Port
				String out1 = "Received Request";
				sendData = out1.getBytes();
				//sets the IP Address to that of this machine
				InetAddress IPAddress1 = InetAddress.getBy-
Name("localhost");
				//makes a new Datagram and sends it
				DatagramPacket sendPacket1 =
						new DatagramPacket(sendData,
sendData.length, IPAddress1, 3021);
				serverSocket.send(sendPacket1);
				//if the second port is present, it sends it to that one.
			}
```

```
		}

		public void run() {
			try {
				this.Run();
			} catch (Exception e) {
				e.printStackTrace();
			}
		}

}
```

End of code Listing

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB IMAL HRA RDRL CIO LL
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
5 (PDF)	DIR USARL RDRL CIH C J CLARKE B HENZ S BROWN RDRL CIN D L MARVEL RDRL CIN T A SWAMI

INTENTIONALLY LEFT BLANK.